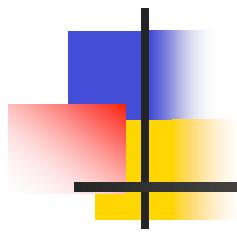
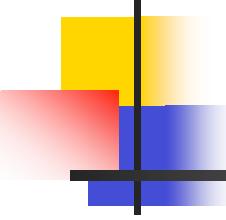


Going parallel using MPI: overview

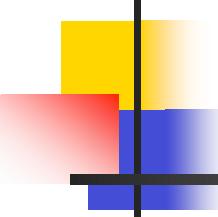


Jelena Marshak
Code 587.0



What will we learn and what not

- Title: Going parallel using MPI
- In this session you will learn:
 - Why to do a parallel computing?
 - Where to start and what is MPI?
 - What does an MPI program looks like?
 - What resources are available at Goddard?
 - How to use them?
- This presentation is NOT
 - A tutorial on learning the MPI

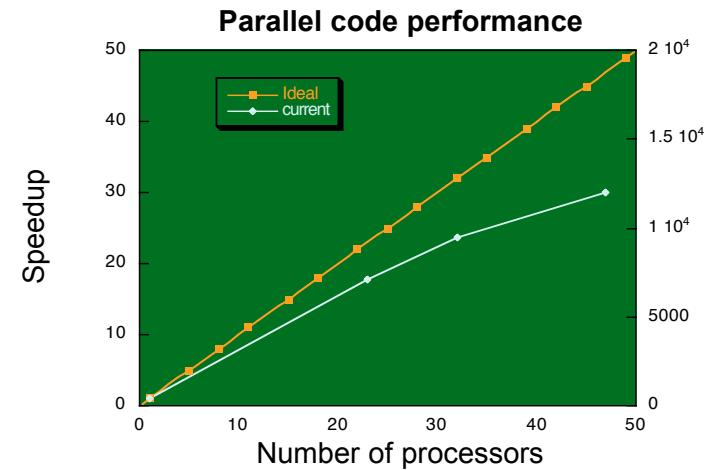


Agenda

- Why to go parallel?
- Thunderhead machine specifications
- Parallel programming
 - Goals, obstacles and challenges
 - Rethinking the problem
 - What to expect?
- Parallel code design
 - Key qualities of the parallel program
 - Where to start?
- Message Passing Interface
 - What is MPI?
 - How to run the MPI program on thunderhead?
- Let's do it together

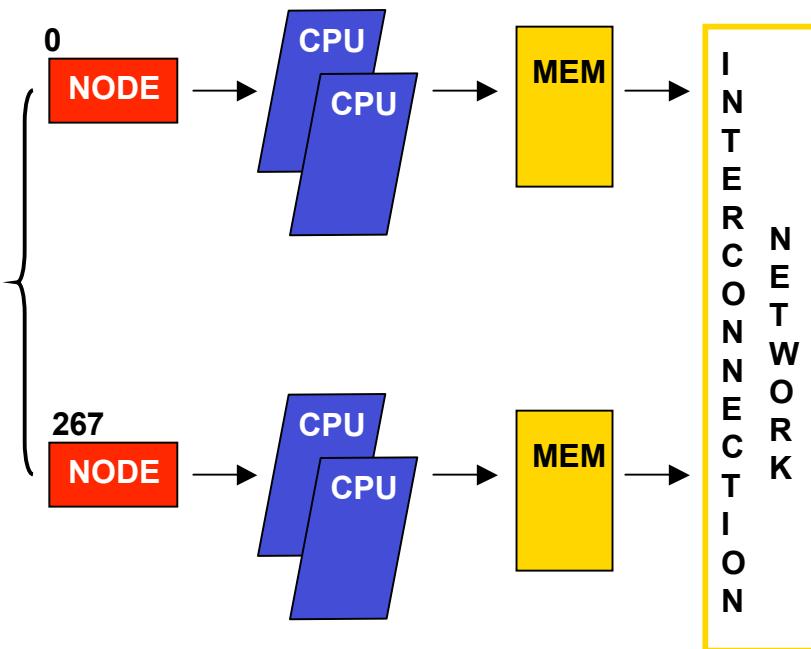
Why parallel computing?

- Price/performance ratio:
 - Cost of computer resources
 - Improved performance - the most compelling reason.
Memory-bound applications can exhibit super linear speedups when ported to distributed memory computers, which typically provide more memory and more cache.

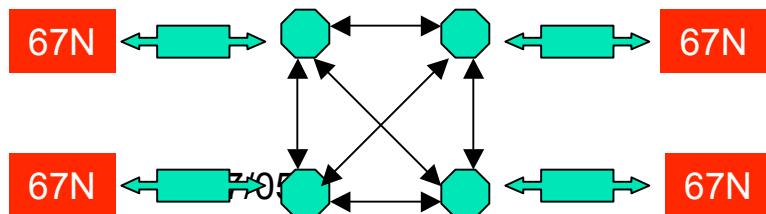


Thunderhead: Specifications

Cluster parallel computational model



Interconnection network topology



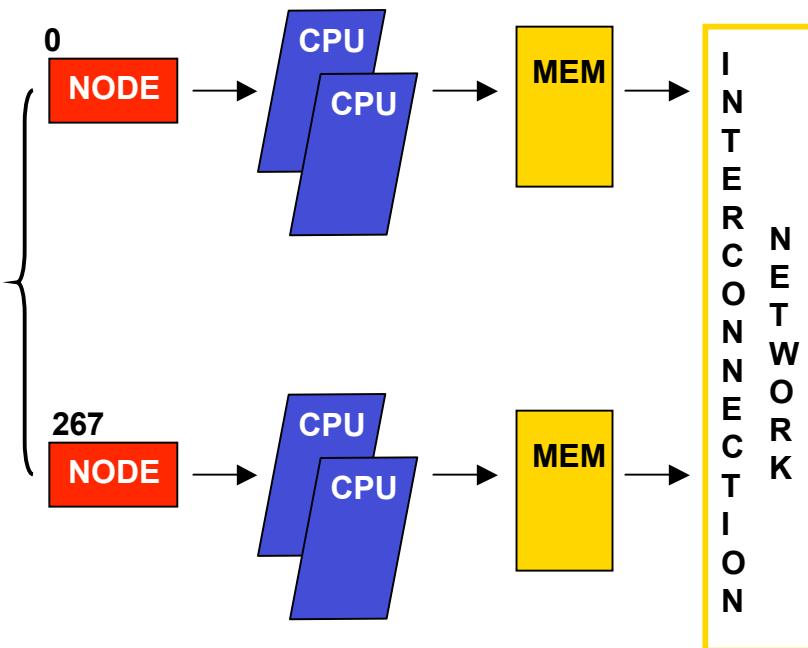
Thunderhead specifications

(from "Thunderhead hardware and Software" by Josephine Palencia)

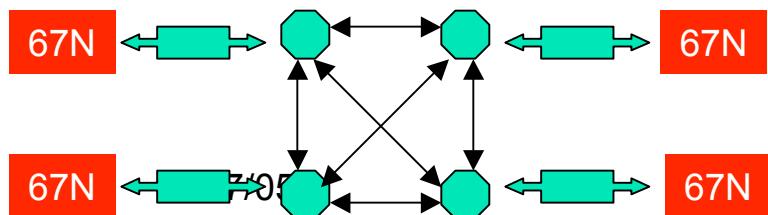
Aggregate Specification	
Number of nodes	268
Total processors	536
Total memory (Gb)	536
Total diskspace (Tbyte))	21.44
Network interconnect 1	Myrinet 2000
Network interconnect 2	Gigabit Ethernet
Network interconnect 3	Fast Ethernet
Theor peak performance (Tflops)	2.5728
Linpack benchmark (Tflops)	1.2
Node Specification	
Motherboard	Tyan Thunder 2720
Number of processors	2X Intel 4 Xeon 2.4Ghz
Memory (Gb)	1024
Local disk (Gb)	80
Network interconnect 1	Myrinet 2000
Network interconnect 3	Fast Ethernet
Network interconnect 2	Gigabit Ethernet
Peak performance (Gflops)	9.6

Thunderhead: Specifications

Cluster parallel computational model



Interconnection network topology



TOP 500 SUPERCOMPUTER SITES

PRESENTED BY
UNIV. OF MANNHEIM
UNIV. OF TENNESSEE
NERSC/LBNL

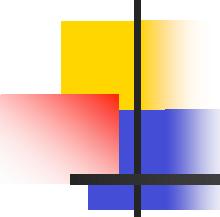
SEARCH FOR:

Pentium4 Xeon 2.4 GHz, Myrinet

NASA/Goddard Space Flight Center
Greenbelt,
United States

Rankings:	Ranking	List	Rmax (GFlops)
326	11/2004	1104	
208	06/2004	1104	
106	11/2003	1104	

System: Pentium4 Xeon 2.4 GHz, Myrinet
 System Type: Pentium4 Xeon 2.4 GHz Cluster - Myrinet
 System Model: NOW Cluster - Intel Pentium - Myrinet
 System Family: NOW - Intel Pentium
 Manufacturer: Self-made
 System URL: <http://newton.gsfc.nasa.gov/>
 Application area:
 Installation Year: 2003



Thunderhead: Meminfo

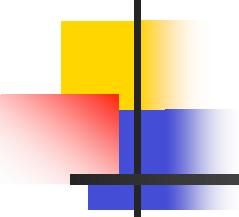
	total:	used:	free:	shared:	buffers:	cached:
Mem:	1056808960	1017782272	39026688	0	80003072	232898560
Swap:	2048344064	115838976	1932505088			
MemTotal:	1032040 kB	(the total amount of memory available)				
MemFree:	38112 kB	(the amount of memory available on the heap; the field you would use as a reference for your allocation amount)				
MemShared:	0 kB	(the amount of shared memory)				
Buffers:	78128 kB	(the amount of memory used by system buffers)				
Cached:	206704 kB	(the amount of memory being cached)				
SwapCached:	20736 kB					
Active:	215680 kB					
Inactive:	132708 kB					
HighTotal:	131008 kB					
HighFree:	10576 kB					
LowTotal:	901032 kB					
LowFree:	27536 kB					
SwapTotal:	2000336 kB	(the amount of swap space)				
SwapFree:	1887212 kB	(the amount of free swap space)				

Cpuinfo

model name: Intel(R) Xeon(TM) CPU 2.40GHz
Cpu speed: 2399.818MHz
Cache (per CPU): 512 KB

Powerbook G4:

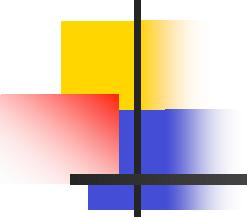
CPU type: PowerPC G4 (1.1)
Cpu speed: 1.5GHz
Cache (per CPU): 512 KB
Mem: 512MB
HD: 70GB
Bus speed: 167MHz



Parallel programming

Goals, obstacles and challenges

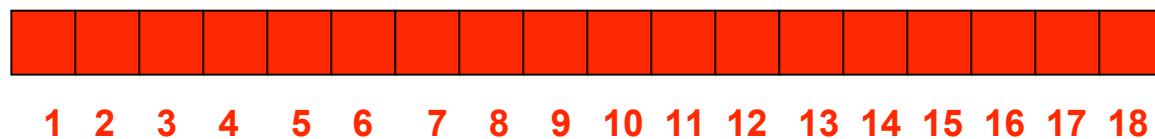
- **Goals** are to reduce the wall clock execution time while keeping the code portable and manageable
 - Ideal: linear speedup and scalability
- **Obstacles** lie in hardware, software and algorithms
 - in hardware - computation and communication speeds are not balanced;
 - in software - limited applicability of compilers that provide automatic parallelization and libraries that work in multiple environments;
 - in algorithms - parallelism becomes a challenge for programmers.
- **Challenge** is to parallelize the algorithm where the apparent serialism exists.

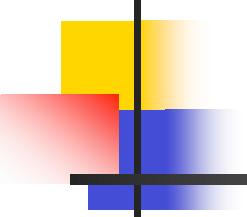


Parallel programming

Rethinking the problem

- Parallelization in the code can come in different ways:
 - from physics (independent physical processes);
 - from mathematics (independent mathematical operations);
 - from software (independent computational tasks).
- May involve rethinking the problem.
- $S = \sum_{i=1}^{18} x_i$, $O(N) = N$ steps

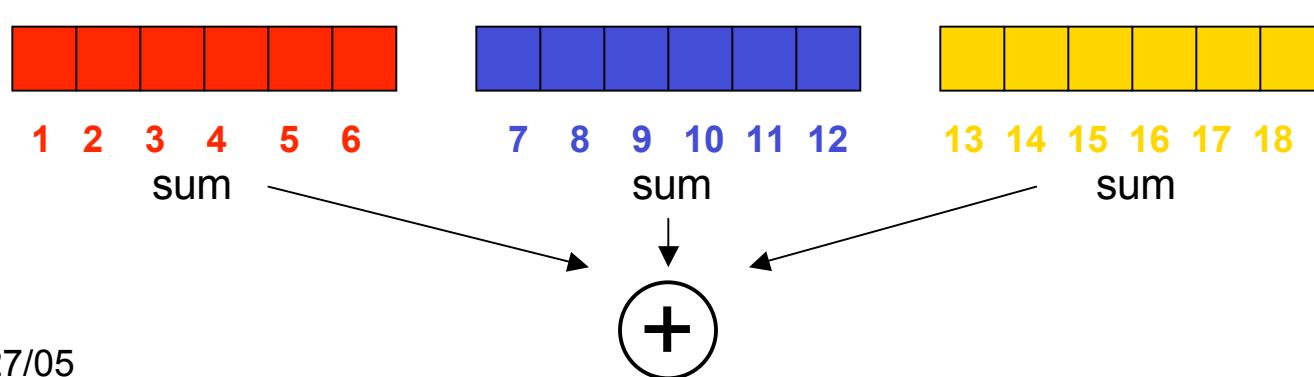


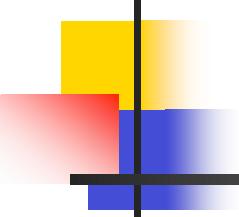


Parallel programming

Rethinking the problem

- Parallelization in the code can come in different ways:
 - from physics (independent physical processes);
 - from mathematics (independent mathematical operations);
 - from software (independent computational tasks).
 - May involve rethinking the problem.
- $S = \sum_{i=1}^{18} x_i = \sum_{i=1}^6 x_i + \sum_{i=7}^{12} x_i + \sum_{i=13}^{18} x_i$ $O(N/k) + \log_2(k) \sim 7$ steps

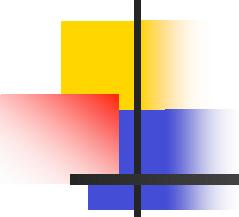




Parallel programming

What to expect?

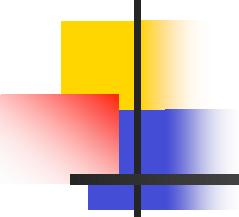
- **Time** - writing a parallel code is a manual time consuming process
 - Analyzing the code for parallelism
 - Rewriting the code
 - Debugging (multiple CPU performing different tasks on different parts of data plus communicating between each other)
- **Cost** - executing a parallel code can be costly process
 - Per CPU time for parallel application is bigger than the one for serial (initialization of parallel work, communication between processors)
 - Total CPU time = $\sum_{i=1}^n \text{cpu}_i$, $n = \text{number_of_all_cpus_used}$
- **Memory use** (some values replicated, some memory used for communication)
- **Portability**



Parallel code design

Parallel code characteristics

- **Parallel execution of tasks**
- Each processor has a **unique bit of work** to do
- Each processor operates on its **own data**
- **Interprocessor communication** is minimal
- **Workload** is balanced between the processors.
- **Synchronization:** more or less?

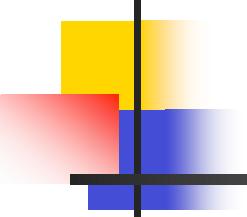


Parallel code design

Where to start?

Major decisions to make:

- Find the tasks in your code that can be executed simultaneously (**functional** or **data parallelism**)
- Define the relationship between the processors - SPMD or Master/Worker



Parallel code design

Where to start? Strategy

- **Functional parallelism**
 - Divide the computation into simultaneously performed tasks - each processor performs a different function or executes a different code section;
 - Appeal: familiar object-oriented approach;
 - Disadvantage: dependence on CPU number, synchronization and communication between tasks (workload and data movement).

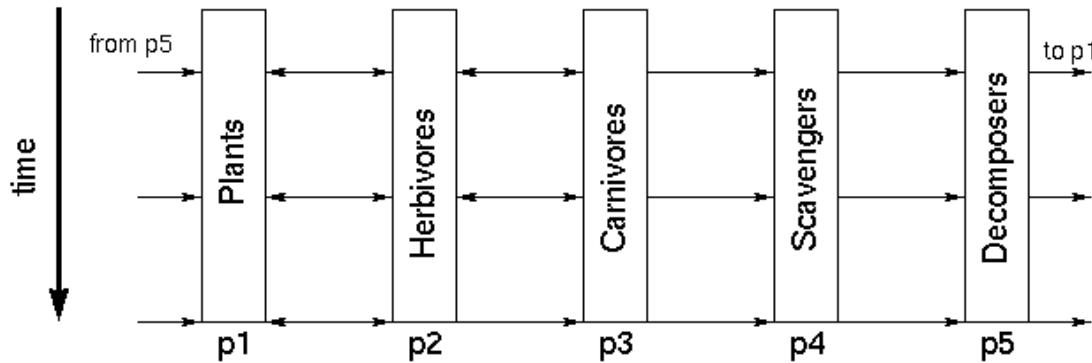
Parallel code design

Where to start? Example #1

Functional parallelism: ecosystem

Courtesy of Cornell Theory Center (CTC)

<http://cmssrv.tc.cornell.edu/CTC-Main/Services/Education/Topics/Parallel/Design/Introduction.htm>



- Calculate the population of the ecosystem;
- 5 tasks: 5 processes
- Load balancing: static (prescheduled).
- The communication pattern: a ring.

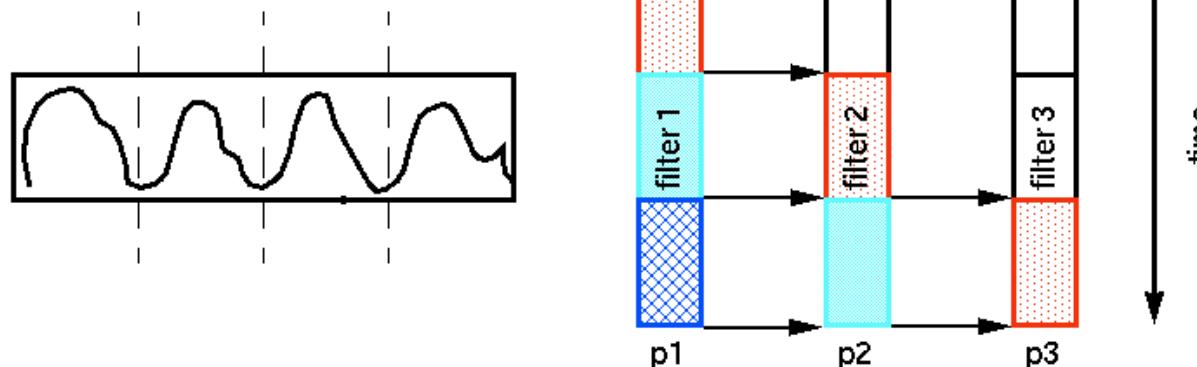
Parallel code design

Where to start? Example # 2

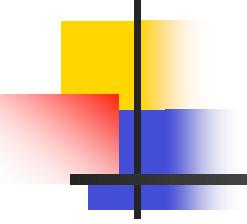
Functional parallelism: audio signal

Courtesy of Cornell Theory Center (CTC)

<http://cmssrv.tc.cornell.edu/CTC-Main/Services/Education/Topics/Parallel/Design/Introduction.htm>



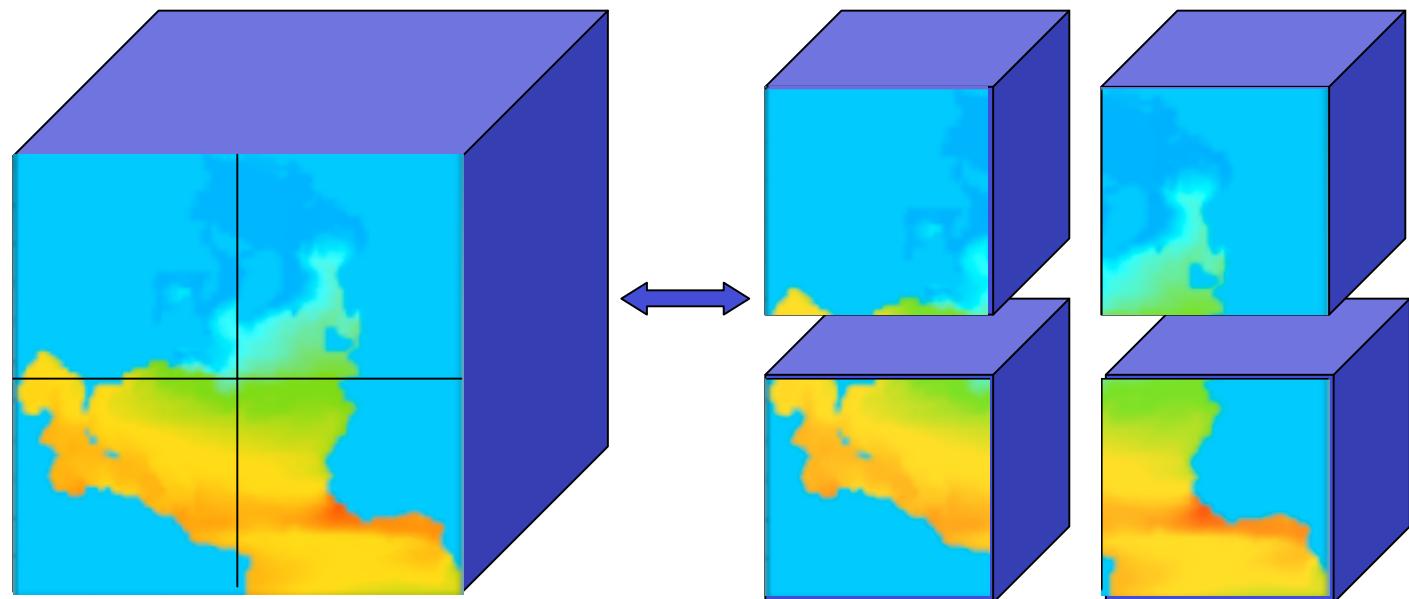
- Process the audio signal; the data set is passed through three distinct computational filters.
- 3 tasks - 3 processes. Each filter is a separate process (task). Load balancing: static (prescheduled).
- The communication pattern is a 1-dimensional mesh. Network topology is important.



Parallel code design

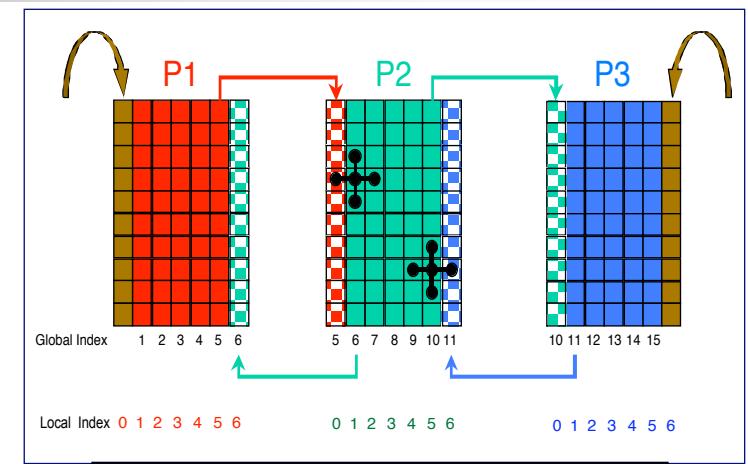
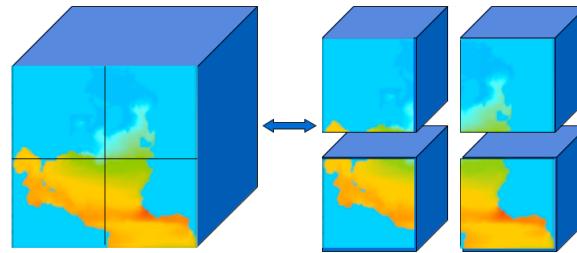
Where to start? Strategy

- Data parallelism
 - Divide data between the processors - each processor performs the same calculation on unique piece of data



Parallel code design

Where to start? Strategy



Courtesy of Dan Schaffer, NOAA FSL

- **Data parallelism**
 - Divide data between the processors
 - Attractiveness
 - scalability and synchronization
 - favorable for applications that require a lot of memory or operate on large grids
 - method takes advantage of large memory, big CPU number and relatively independent data
 - Disadvantage
 - data dependencies and communication/neighborhood problem

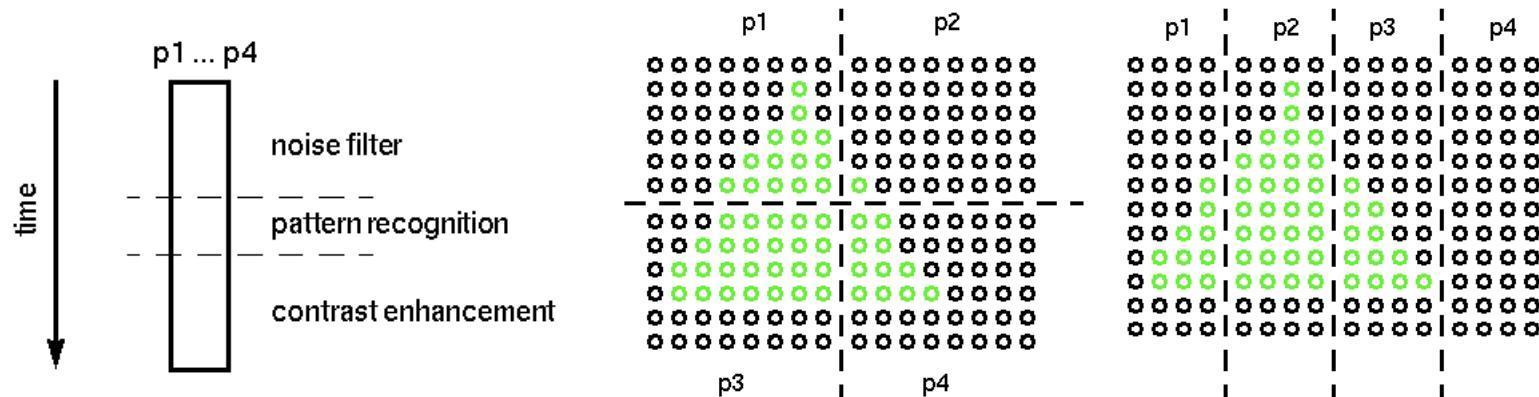
Parallel code design

Where to start? Example # 3

Data parallelism: image processing

Courtesy of Cornell Theory Center (CTC)

<http://cmssrv.tc.cornell.edu/CTC-Main/Services/Education/Topics/Parallel/Design/Introduction.htm>



- 4 processes (dash lines on the left image represent barriers);
- Load balancing: static (prescheduled);
- 1D and 2D data decomposition;

Parallel code design

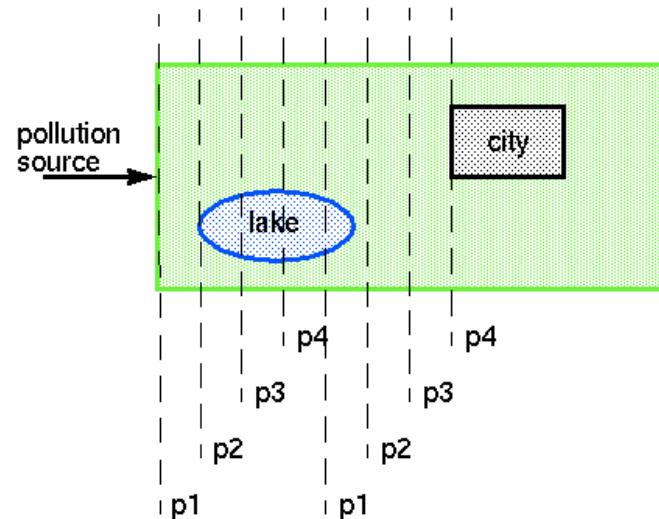
Where to start? Example # 4

Data parallelism: pollution

Courtesy of Cornell Theory Center (CTC)

<http://cmssrv.tc.cornell.edu/CTC-Main/Services/Education/Topics/Parallel/Design/Introduction.htm>

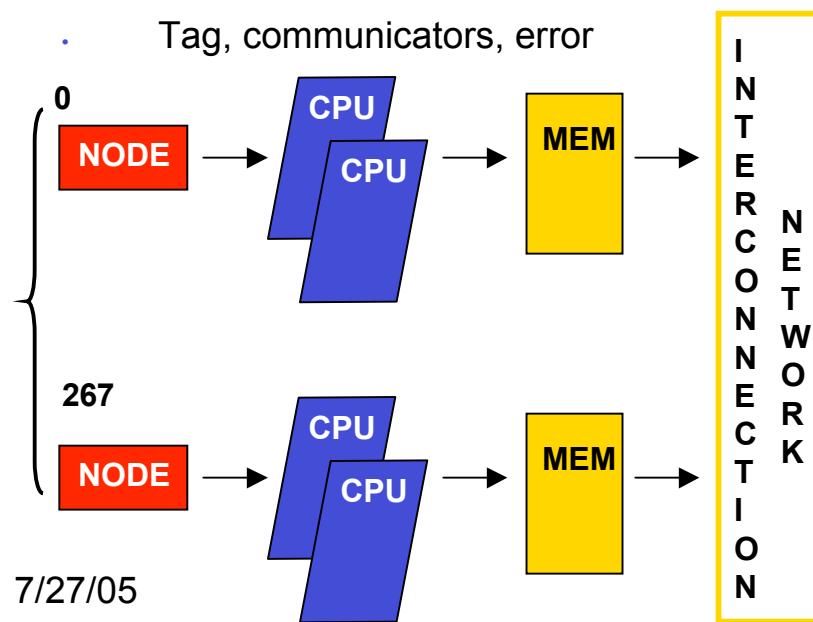
- Calculate the effect of pollution on tree growth and mortality for a geographic area;
- 4 processes;
- Load balancing: cyclic decomposition;
- Domain: irregular grid, dynamic;
- No interaction between members.



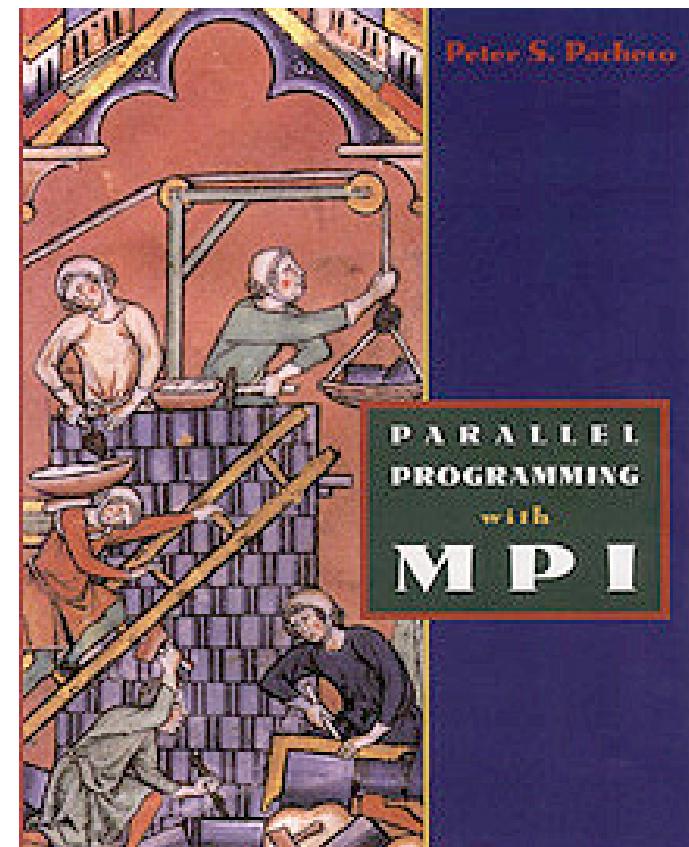
Message Passing Interface

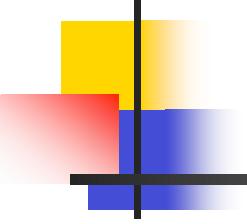
What is MPI? Mechanism

- A message
 - MPI addresses the message-passing model
 - Basic mechanism - exchange of messages
 - Message and its attributes
 - Data to be passed
 - Data type and size
 - Destination
 - Tag, communicators, error



7/27/05





Message Passing Interface

What is MPI? Scope

- **MPI is a library, not a language**
 - Standardized in 1994 by MPI Forum, represented by Convex, Cray, IBM, Intel, Meiko, nCUBE, NEC, and Thinking Machines
 - MPI goal: to provide portable, efficient and functional parallel code
 - MPI specifies the names, calling sequences, and results of subroutines (Fortran) and functions (C)
- **Is MPI large or small?**
 - MPI is large (125 functions)
 - MPI is small (6 functions)

code.f:

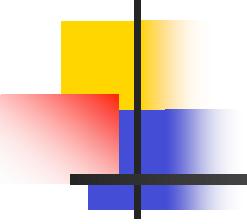
call MPI_INIT(ierr)

code.c:

ierr=MPI_Init(int
*argc, char ***argv)

F90 code.f -Impi

CC code.c -Impi



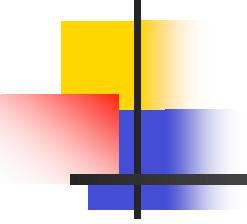
Message Passing Interface

What is MPI? Header and format

- **MPI header file**
 - Fortran:
include "mpif.h"
 - C:
#include "mpi.h"
- **MPI format**
 - Fortran:
Call MPI_NAME_OF_FUNCTION(parameters,error)
 - C:
error=MPI_Name_of_function(parameters)

Required functions:

```
MPI_INIT  
MPI_COMM_SIZE  
MPI_COMM_RANK  
MPI_SEND  
MPI_RECV  
MPI_FINALIZE
```



Message Passing Interface

What is MPI? Start and finish

- **MPI initialization**

- Must be the 1-st MPI call
- Establishes the MPI environment
- One per executable

Fortran:

```
call MPI_INIT(ierr)
```

C:

```
ierr=MPI_Init(int *argc, char ***argv)
```

Required functions:

MPI_INIT

MPI_COMM_SIZE

MPI_COMM_RANK

MPI_SEND

MPI_RECV

MPI_FINALIZE

- **MPI termination**

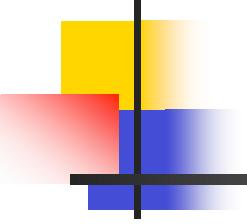
- Must be the last MPI call
- Terminates the MPI environment
- Must be made by every processor

Fortran:

```
call MPI_FINALIZE(ierr)
```

C:

```
ierr=MPI_Finalize()
```



Message Passing Interface

What is MPI? Size and Rank

- **MPI size**

- returns the number of processes the user started
- `comm` defines the group of processes (system or user)
- `MPI_COMM_WORLD` - default `communicator`

Fortran:

```
call MPI_COMM_SIZE(comm,num_proc,ierr)
```

C:

```
ierr=MPI_Comm_size(MPI_Comm comm, int *num_proc)
```

- **MPI rank**

- returns the current processor number
- `rank` is the processor number within communicator
- rank (0 - size-1) is not the PE number

Fortran:

```
call MPI_COMM_RANK(comm,rank,ierr)
```

C:

```
ierr=MPI_Comm_rank(MPI_Comm comm, int *rank)
```

Required functions:

`MPI_INIT`

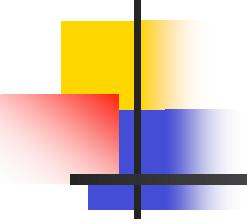
`MPI_COMM_SIZE`

`MPI_COMM_RANK`

`MPI_SEND`

`MPI_RECV`

`MPI_FINALIZE`



Message Passing Interface

What is MPI? Send and Receive

- **MPI send**

- Standard blocking send
- `buf, count, datatype, dest, tag`

Fortran:

```
call MPI_SEND(buf, count, datatype, dest, tag, comm, ierr)
```

C:

```
ierr=MPI_Send(void *buf, int count, MPI_datatype datatype,  
int dest, int tag, MPI_Comm comm)
```

- **MPI receive**

- Standard blocking receive
- `source, status`

Fortran:

```
call MPI_RECV(buf, count, datatype, source, tag, comm, status, ierr)
```

C:

```
ierr=MPI_Recv(void *buf, int count, MPI_datatype datatype,  
int source, int tag, MPI_Comm comm, MPI_status *status)
```

Required functions:

`MPI_INIT`

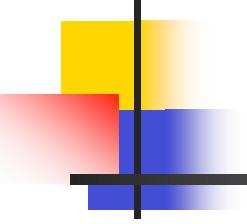
`MPI_COMM_SIZE`

`MPI_COMM_RANK`

`MPI_SEND`

`MPI_RECV`

`MPI_FINALIZE`



Message Passing Interface

What is MPI? Send and Receive

- **MPI send**

- Standard blocking send
- **buf, count, datatype, dest, tag**

Fortran:

```
call MPI_SEND(buf, count, datatype, dest, tag, comm, ierr)
```

C:

```
ierr=MPI_Send(void *buf, int count, MPI_datatype datatype,  
int dest, int tag, MPI_Comm comm)
```

Some MPI Datatypes:

MPI_INTEGER	Fortran
-------------	---------

MPI_REAL	
MPI_DOUBLE_PRECISION	
MPI_COMPLEX	
MPI_CHARACTER	
MPI_BYTE	
MPI_LOGICAL	

MPI_INT	C
---------	---

MPI_FLOAT	
MPI_DOUBLE	
MPI_SHORT, MPI_LONG	
MPI_CHAR	
MPI_BYTE	
MPI_UNSIGNED	

- **MPI receive**

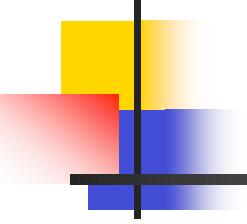
- Standard blocking receive
- **source, status**

Fortran:

```
call MPI_RECV(buf, count, datatype, source, tag, comm, status, ierr)
```

C:

```
ierr=MPI_Recv(void *buf, int count, MPI_datatype datatype,  
int source, int tag, MPI_Comm comm, MPI_Status *status)
```



Message Passing Interface

What is MPI? More

More functions:

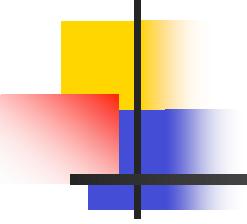
- MPI_ALLREDUCE
- MPI_BCAST
- MPI_COMM_GROUP
- MPI_GROUP_INCL
- MPI_COMM_CREATE
- MPI_GROUP_FREE
- MPI_BARRIER
- MPI_WAIT
- MPI_TYPE_VECTOR
- MPI_IRecv
- MPI_ISend

7/27/05

- MPI_INIT
- MPI_COMM_SIZE
- MPI_COMM_RANK
- MPI_SEND
- MPI_RECV
- MPI_FINALIZE

Required functions:

```
call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,pe,ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,npes,ierr)
...
call MPI_FINALIZE(ierr)
```



Message Passing Interface

What is MPI? More

More functions:

MPI_ALLREDUCE

MPI_BCAST

MPI_COMM_GROUP

MPI_GROUP_INCL

MPI_COMM_CREATE

MPI_GROUP_FREE

MPI_BARRIER

MPI_WAIT

MPI_TYPE_VECTOR

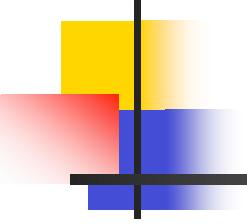
MPI_IRecv

MPI_ISEND

```
call MPI_SEND(xglobal,1,newtype,  
& receiver, receiver,MPI_COMM_WORLD, ierr)
```

c number of blocks
c number of elements in each block
c stride or spacing between start of each block
c

```
call MPI_TYPE_VECTOR(10,1,1,  
& MPI_DOUBLE_PRECISION,newtype,ierr)  
call MPI_TYPE_COMMIT(newtype,ierr)  
...  
call MPI_TYPE_FREE(newtype,ierr)
```



Message Passing Interface

What is MPI? More

More functions:

MPI_ALLREDUCE

MPI_BCAST

MPI_COMM_GROUP

MPI_GROUP_INCL

MPI_COMM_CREATE

MPI_GROUP_FREE

MPI_BARRIER

MPI_WAIT

MPI_TYPE_VECTOR

MPI_IRecv

MPI_ISEND

7/27/05

```
subroutine gdsum(x,n,work)
integer n,ierr
real*8 x(n),work(n)
include 'mpif.h'
```

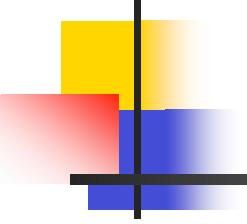
```
call gdsum(data,2000,work)
```

```
call MPI_ALLREDUCE(x,work,n,MPI_DOUBLE_PRECISION,
&                  MPI_SUM, MPI_COMM_WORLD, ierr)
return
end
```

```
subroutine bcast(x,n,src)
integer n,src,ierr
real*8 x(n)
include 'mpif.h'
```

```
call bcast(data,2000,0)
```

```
call MPI_BCAST(x,n,MPI_DOUBLE_PRECISION,src,
&                  MPI_COMM_WORLD,ierr)
return
end
```



Message Passing Interface

What is MPI? More

More functions:

MPI_ALLREDUCE

MPI_BCAST

MPI_COMM_GROUP

MPI_GROUP_INCL

MPI_COMM_CREATE

MPI_GROUP_FREE

MPI_BARRIER

MPI_WAIT

MPI_TYPE_VECTOR

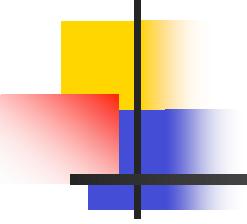
MPI_IRecv

MPI_Isend

C Send to the left, receive from right
call MPI_ISEND(a,1,vtype,
& left,101,MPI_COMM_ICE,rhandle,ierr)
call MPI_RECV(a,1,vtype,
& right,101,MPI_COMM_ICE,istatus,ierr)

C wait for non-blocking send to complete
call MPI_WAIT(rhandle,istatus,ierr)

...
call MPI_BARRIER(MPI_COMM_ICE,ierr)



Message Passing Interface

What is MPI? More

More functions:

MPI_ALLREDUCE

MPI_BCAST

MPI_COMM_GROUP

MPI_GROUP_INCL

MPI_COMM_CREATE

MPI_GROUP_FREE

MPI_BARRIER

MPI_WAIT

MPI_TYPE_VECTOR

MPI_IRecv

MPI_ISEND

```
call create_new_group(ranks,n_pes_ice,MPI_COMM_ICE)
```

c Create the new group

```
subroutine create_new_group(rranks,n,newcomm)
```

```
implicit none
```

```
include 'mpif.h'
```

```
integer n,rranks(n)
```

```
integer newcomm
```

```
integer world_group,new_group,ierr
```

```
call MPI_COMM_GROUP(MPI_COMM_WORLD,world_group,ierr)
```

```
call MPI_GROUP_INCL(world_group,n,rranks,new_group,ierr)
```

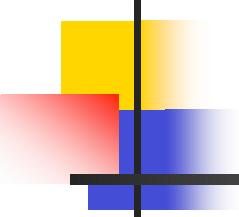
```
call MPI_COMM_CREATE(MPI_COMM_WORLD,new_group,newcomm,ierr)
```

```
call MPI_GROUP_FREE(new_group,ierr)
```

```
call MPI_GROUP_FREE(world_group,ierr)
```

```
return
```

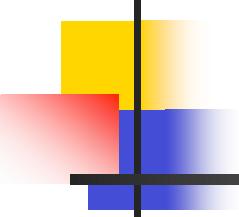
```
end
```



How to run the MPI program

Where to start on Thunderhead?

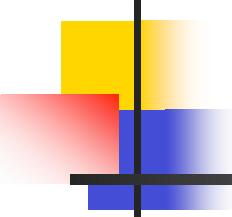
- Read ‘[User’s Guide](#)’ on Thunderhead WEB site: <http://thunderhead.gsfc.nasa.gov/>
- Follow the link to MPI on the same page
- Try the examples of MPI code
(They can be found on Thunderhead machine in your home directory: \$HOME/.mpi_gm/mpich_home/examples)



How to run the MPI program

Compiling and executing the job

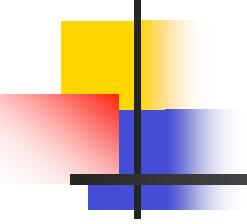
- Libraries (mpi_env)
 - gnu build, (gnu_io build) - for gcc and g77 compiler (with mpi-io support)
 - intel build, (intel_io build) - for intel 8.0 compilers (with mpi-io support)
 - intel_old build - for intel 7.0 compilers
 - lahey build,(lahey_io build) - for lahey fortran compiler (with mpi-io support)
 - pgi build, (pgi_io build) - for portland group compilers (with mpi-io support)
 - nag build, (nag_io build) - for nag fortran compiler (with mpi-io support)
 - nag_debug build
- Compilers
 - mpif90, mpif77,mpicc, mpiCC
- Scheduler commands
 - Ltmsuper, Ltmbegin, Ltmend, Ltmterminate, Ltmpi, Ltmstate
- http://thunderhead.gsfc.nasa.gov/Users_Guide



How to run the MPI program

Compiling and executing the job

- Example of standard procedure:
 - `pwd (/home/wly0m/myDir)`
 - `mpi_env -p`
 - `mpi_env -c nag`
 - `mpif90 code.f -o code.x`
 - `cp -p code.x $HOME/rhrome/myDir`
 - `ltmsuper`
 - `ltmbegin -n number_of_nodes`
 - `ltmpi code.x &`
 - `ltmend`
 - `ltmterminate`



Let's do it together

Steps

Walk through example is the Courtesy of Cornell Theory Center (CTC)
<http://cmssrv.tc.comell.edu/CTC-Main/Services/Education/Topics/Parallel/Design/introduction.htm>

1. Define the problem
2. Choose the strategy and workload
3. Define the relationship between the processes
4. Design the algorithm
5. Write pseudo code
6. Write an MPI code
7. Run it on thunderhead
8. Analyze the results

Let's do it together

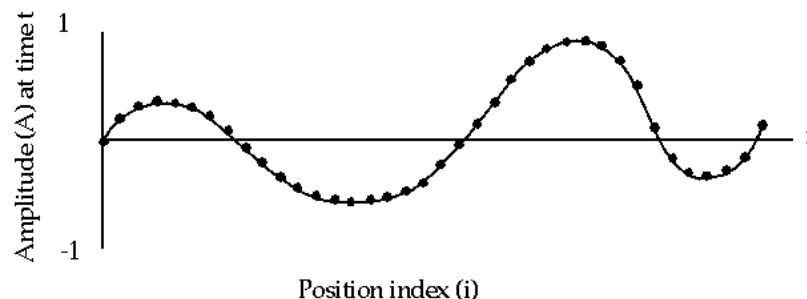
Step # 1: define the problem

Problem: Calculate the amplitude along a uniform, vibrating string after a specified amount of time has elapsed.



The equation to be solved is the finite-difference approximation to the 1D wave equation:

$$A(i,t+1) = (2.0 * A(i,t)) - A(i,t-1) + (\text{const} * (A(i-1,t) - (2.0 * A(i,t)) + A(i+1,t)))$$



Let's do it together

Step # 2: strategy and domain

- 1) Function or data decomposition?

$A(i,t+1) = (2.0 * A(i,t)) - A(i,t-1) + (\text{const} * (A(i-1,t) - (2.0 * A(i,t)) + A(i+1,t)))$  Data

- 2) How we will define the data domain for each process? Time or position? Which can be calculated concurrently? Check for data dependency:

$A(l,t+1)$ depends on $A(l,t)$ & $A(l,t-1)$; $A(l,t+1)$ requires $A(l-1,t)$ & $A(l+1,t)$  Position

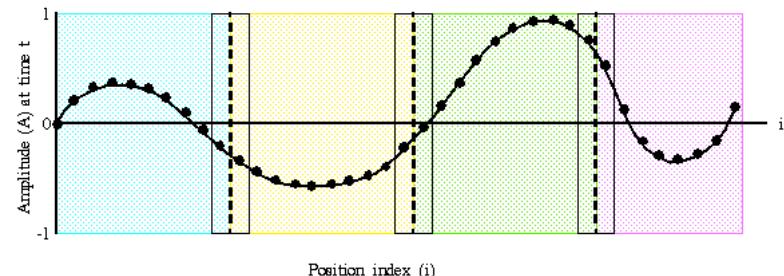
- 3) Define the workload for data decomposition

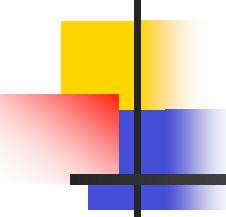
Data replication

Load balancing (cyclic or block)

Communication (cyclic or block)

 Block decomposition by Position





Let's do it together

Step # 3: SPMD and Master/Worker

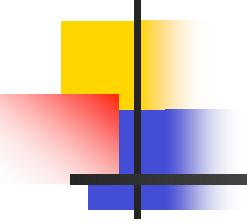
Two solutions to the problem: SPMD and Master/Worker

SPMD:

- SPMD: all processes run the same program
- DATA PARALLEL: the work is partitioned by data
- SCALABLE: none of the work is constrained by one process and no global communication is required

SPMD with Master/Worker embedded:

- SPMD: all processes run the same program
- DATA PARALLEL: the work is partitioned by data
- MASTER WORKER: flow control assigns certain work to a "Master" or "Worker"



Let's do it together

Step # 4: code structure

1. Read in starting values
2. Establish communication channels
3. Divide data among processes
4. Exchange endpoints
5. Calculate amplitude for new time step
6. Repeat steps 4 and 5 for given number of time steps
7. Output results

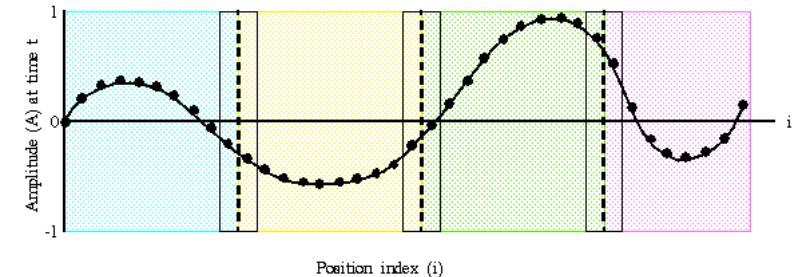
Let's do it together

Step # 5: SMPD pseudocode

```

C Learn number of tasks and taskid
call initialize task
call get task identification and information
C Identify left and right neighbors
C Get program parameters
read tpoints, nsteps
C Divide data amongst processes
read values
C Update values for each point along string
do t = 1, nsteps
C   Send to left, receive from right
    call send left endpoint to left neighbor
    call receive left endpoint from right neighbor
C   Send to right, receive from left
    call send right endpoint to right neighbor
    call receive right endpoint from left neighbor
C   Update points along line
    do i = 1, npoints
      newval(i) = (2.0 * values(i)) - oldval(i) + (sqtau * (values(i-1) - (2.0 * values(i)) + values(i+1)))
    end do
  end do
C Write results out to file
write values
call terminate parallel environment

```



```

C Learn number of tasks and taskid
call mpi_init(ierr)
call mpi_comm_rank(MPI_COMM_WORLD, taskid, ierr)
call mpi_comm_size(MPI_COMM_WORLD, nproc, ierr)

```

$$\text{newval}(i) = (2.0 * \text{values}(i)) - \text{oldval}(i) + (\text{sqtau} * (\text{values}(i-1) - (2.0 * \text{values}(i)) + \text{values}(i+1)))$$

end do

end do

C Write results out to file

write values

call **terminate parallel environment**

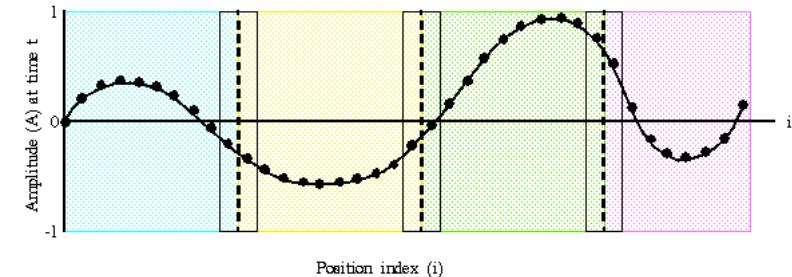
Let's do it together

Step # 5: SMPD pseudocode

```

C Learn number of tasks and taskid
call initialize task
call get task identification and information
C Identify left and right neighbors
C Get program parameters
read tpoints, nsteps
C Divide data amongst processes
read values
C Update values for each point along string
do t = 1, nsteps
    C Send to left, receive from right
    call send left endpoint to left neighbor
    call receive left endpoint from right neighbor
    C Send to right, receive from left
    call send right endpoint to right neighbor
    call receive right endpoint from left neighbor
    C Update points along line
    do i = 1, npoints
        newval(i) = (2.0 * values(i)) - oldval(i) + (sqtau * (values(i-1) - (2.0 * values(i)) + values(i+1)))
        end do
    end do
C Write results out to file
write values
call terminate parallel environment

```



```

C Communication will not wrap, so end nodes have "null" neighbors
if (taskid .eq. nproc-1) then
    right = MPI_PROC_NULL
else
    right = taskid + 1
end if

```

$$\text{newval}(i) = (2.0 * \text{values}(i)) - \text{oldval}(i) + (\text{sqtau} * (\text{values}(i-1) - (2.0 * \text{values}(i)) + \text{values}(i+1)))$$

end do

end do

C Write results out to file

write values

call **terminate parallel environment**

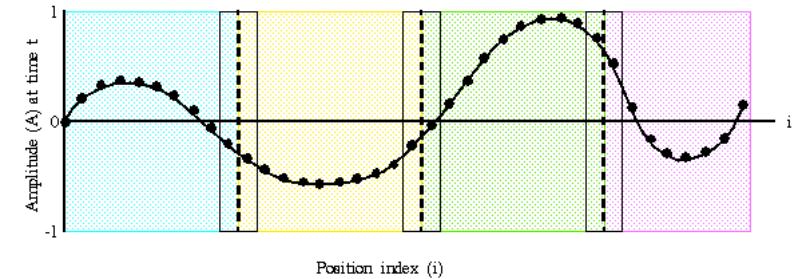
Let's do it together

Step # 5: SMPD pseudocode

```

C Learn number of tasks and taskid
call initialize task
call get task identification and information
C Identify left and right neighbors
C Get program parameters
read tpoints, nsteps
C Divide data amongst processes
read values
C Update values for each point along string
do t = 1, nsteps
C   Send to left, receive from right
    call send left endpoint to left neighbor
    call receive left endpoint from right neighbor
C   Send to right, receive from left
    call send right endpoint to right neighbor
    call receive right endpoint from left neighbor
C   Update points along line
    do i = 1, npoints
      newval(i) = (2.0 * values(i)) - oldval(i) + (sqtau * (values(i-1) - (2.0 * values(i)) + values(i+1)))
    end do
  end do
C Write results out to file
write values
call terminate parallel environment

```



```

C End nodes have "null" neighbors -- this prevents communication
from wrapping
call mpi_isend(values(1), 1, MPI_DOUBLE_PRECISION, left,
.   E_RtoL, MPI_COMM_WORLD, request, ierr)
call mpi_recv(values(npoin+1), 1, MPI_DOUBLE_PRECISION,
.   right, E_RtoL, MPI_COMM_WORLD, status, ierr)
call mpi_wait(request, status, ierr)

```

$$\text{newval}(i) = (2.0 * \text{values}(i)) - \text{oldval}(i) + (\text{sqtau} * (\text{values}(i-1) - (2.0 * \text{values}(i)) + \text{values}(i+1)))$$

end do

end do

C Write results out to file

write values

call **terminate parallel environment**

Let's do it together

Step # 5: SMPD pseudocode

C Learn number of tasks and taskid
 call **initialize task**
 call **get task identification and information**

C Identify left and right neighbors

C Get program parameters
 read tpoints, nsteps

C Divide data amongst processes
 read values

C Update values for each point along string
do t = 1, nsteps

C Send to left, receive from right
 call **send left endpoint to left neighbor**
 call **receive left endpoint from right neighbor**

C Send to right, receive from left
 call **send right endpoint to right neighbor**
 call **receive right endpoint from left neighbor**

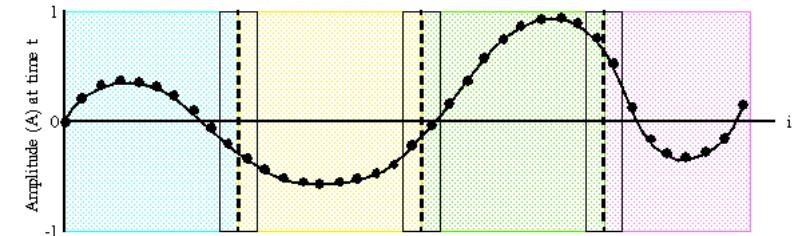
C Update points along line
do i = 1, npoints

```
    newval(i) = (2.0 * values(i)) - oldval(i) + (sqtau * (values(i-1) - (2.0 * values(i)) + values(i+1)))
```

end do

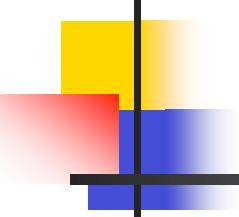
end do

C Write results out to file
 write values
 call **terminate parallel environment**



```
open(unit=12, access='direct',
+     file='/data1/wly0m/output.data', recl=16)
j = 1
do i = first, first + npoints - 1
    write(unit=12,rec=i) values(j)
    j = j + 1
end do
write(unit=12,rec=tpoints+1) tpoints
close(unit=12)
```

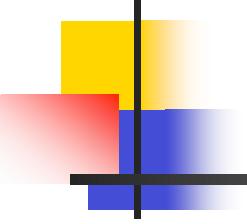
call mpi_finalize(ierr)



Let's do it together

Step # 6: MPI code printout 1

```
[wly0m@thunder1 first]$ cat wave.f | more
C -----
C This program implements the concurrent wave equation described
C in Chapter 5 of Fox et al., 1988, Solving Problems on Concurrent
C Processors, vol 1.
C
C A vibrating string is decomposed into points. Each processor is
C responsible for updating the amplitude of a number of points over
C time.
C
C At each iteration, each processor exchanges boundary points with
C nearest neighbors. This version uses low level sends and receives
C to exchange boundary points.
C
C AUTHOR: Roslyn Leibensperger (C program for MPL)
C REVISED: 06/07/93 R. Leibensperger Ported to Fortran
C CONVERTED to MPI: 11/12/94 by Xianneng Shen
C MODIFIED for Design talk 10/95 by Susan Mehringer
C IMPROVEMENT TO ENDPOINT EXCHANGE: 8/30 R. Leibensperger
C -----
C -----
```



Let's do it together

Step # 6: MPI code printout 2

C Explanation of constants and variables used in common blocks and

C include files

C MASTER = task ID of master

C INTSIZE = size of integer in bytes

C REAL8SIZE = size of real*8 in bytes

C E_OUT1, E_OUT2 = message types

C taskid = task ID

C nproc = number of tasks

C tpoints = total points along wave

C nsteps = number of time steps

C npoints = number of points handled by this task

C first = index of first point handled by this task

C values(0:1001) = values at time t

C oldval(0:1001) = values at time (t-dt)

C newval(0:1001) = values at time (t+dt)

C -----

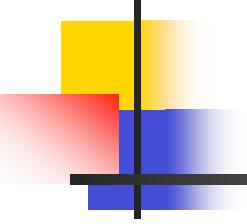
```
program wave_spmd
implicit none
include 'mpif.h'
c include 'parameters.h'
integer ierr
integer taskid, nproc
common/config/taskid, nproc

integer tpoints, nsteps
common/inputs/tpoints, nsteps

integer npoints, first
common/decomp/npoints, first

real*8 values(0:1001), oldval(0:1001), newval(0:1001)
common/data/values, oldval, newval

integer i, j, left, right, nbuf(4)
```



Let's do it together

Step # 6: MPI code printout 3

C Learn number of tasks and taskid

```
call mpi_init(ierr)
call mpi_comm_rank(MPI_COMM_WORLD, taskid, ierr)
call mpi_comm_size(MPI_COMM_WORLD, nproc, ierr)
write (*,5) taskid
5 format (I5, ': Wave Program running')
```

C Determine left and right neighbors

```
C Communication will not wrap, so end nodes have "null"
neighbors
if (taskid .eq. nproc-1) then
    right = MPI_PROC_NULL
else
    right = taskid + 1
end if

if (taskid .eq. 0) then
    left = MPI_PROC_NULL
else
    left = taskid - 1
end if
```

C Get program parameters and initial wave values

```
tpoints=1000
nsteps=100
write (*,10) taskid, tpoints, nsteps
10 format(I5, ': points = ', I5, ' steps = ', I5)
call init_line
```

C Update values along the wave for nstep time steps

```
call update(left, right)
```

C Write results out to file

```
open(unit=12, access='direct',
+     file='/data1/wly0m/output.data', recl=16)
j = 1
do i = first, first + npoints - 1
    write(unit=12,rec=i) values(j)
    j = j + 1
end do
write(unit=12,rec=tpoints+1) tpoints
close(unit=12)
```

C Finalize

```
call mpi_finalize(ierr)
end
```

Let's do it together

Step # 6: MPI code printout 4

C Read in initial points on line

subroutine init_line

implicit none

integer taskid, nproc
common/config/taskid, nproc

integer tpoints, nsteps
common/inputs/tpoints, nsteps

integer npoints, first
common/decomp/npoints, first

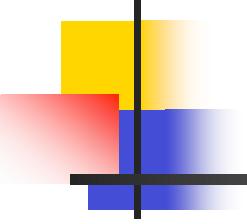
real*8 values(0:1001), oldval(0:1001),
newval(0:1001)
common/data/values, oldval, newval

integer nmin, nleft, npts, i, j, k

nmin = tpoints/nproc
nleft = mod(tpoints, nproc)

```
k = 0
do i = 0, nproc-1
    if (i .lt. nleft) then
        npts = nmin + 1
    else
        npts = nmin
    endif
    if (taskid .eq. i) then
        first = k + 1
        npoints = npts
        write (*,15) taskid, first, npts
15      format (I5, 'IL: first = ', I5, ' npoints = ', I5)
        do j = 1, npts
            k=k+1
            values(j)=sin(6.14*k/tpoints)
        end do
    else
        k = k + npts
    end if
end do

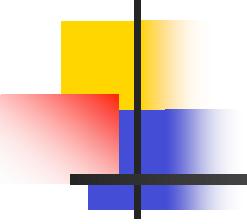
do i = 1, npoints
    oldval(i) = values(i)
end do
end
```



Let's do it together

Step # 6: MPI code printout 5

```
C -----
C   Calculate new values using wave equation
C -----  
  
subroutine do_math(i)
implicit none
integer i  
  
integer tpoints, nsteps
common/inputs/tpoints, nsteps  
  
real*8 values(0:1001), oldval(0:1001), newval(0:1001)
common/data/values, oldval, newval  
  
real*8 dtime, c, dx, tau, sqtau  
  
dtime = 0.3
c = 1.0
dx = 1.0
tau = (c * dtime / dx)
sqtau = tau * tau
newval(i) = (2.0 * values(i)) - oldval(i)
&   + (sqtau * (values(i-1) - (2.0 * values(i)) + values(i+1)))
end
```



Let's do it together

Step # 6: MPI code printout 6

C -----
C Update all values along line a specified number of
times

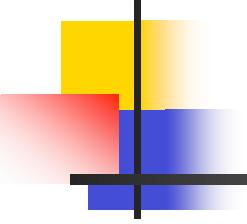
C -----
subroutine update(left, right)
implicit none
include 'mpif.h'
integer left, right
integer ierr, status(MPI_STATUS_SIZE), request
c include 'parameters.h'

integer npoints, first
common/decomp/npoints, first

integer tpoints, nsteps
common/inputs/tpoints, nsteps

real*8 values(0:1001), oldval(0:1001), newval(0:1001)
common/data/values, oldval, newval

integer E_Rtol, E_LtoR
parameter (E_Rtol = 10)
parameter (E_LtoR = 20)
integer i, j, id_rtol, id_ltor, nbytes, msglen



Let's do it together

Step # 6: MPI code printout 7

C Update values for each point along string
do i = 1, nsteps

C Send to the left, receive from the right
C End nodes have "null" neighbors -- this prevents communication
C from wrapping
call **mpi_isend**(values(1), 1,
MPI_DOUBLE_PRECISION, left,
E_RtoL, MPI_COMM_WORLD, request, ierr)
call **mpi_recv**(values(npoin+1), 1,
MPI_DOUBLE_PRECISION,
right, E_RtoL, MPI_COMM_WORLD, status, ierr)
call **mpi_wait**(request, status, ierr)

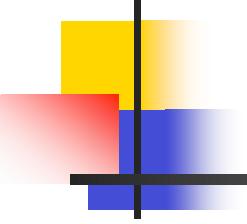
C Send to the right, receive from the left
call **mpi_isend**(values(npoin), 1,
MPI_DOUBLE_PRECISION,
right, E_LtoR, MPI_COMM_WORLD, request, ierr)
call **mpi_recv**(values(0), 1,
MPI_DOUBLE_PRECISION, left,
E_LtoR, MPI_COMM_WORLD, status, ierr)
call **mpi_wait**(request, status, ierr)

C Update points along line
do j = 1, npoints
Global endpoints
if ((first+j-1 .eq. 1).or.(first+j-1 .eq. tpoints))then
newval(j) = 0.0
else
call **do_math**(j)
end if
end do

do j = 1, npoints
oldval(j) = values(j)
values(j) = newval(j)
end do
call **MPI_BARRIER**(MPI_COMM_WORLD,ierr)

end do

end



Let's do it together

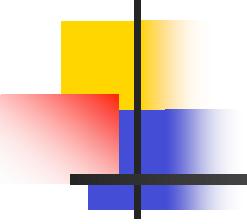
Step # 7: Execute procedure

```
[wly0m@thunder1 first]$ cat rwave
ltmbegin -n 2

##### COMPILE PROGRAM:
rm prog.x
mpif90 -o prog.x -maxcontin=25 -w -V wave.f

##### RUN
cp -p prog.x $HOME/rhome/first
date
ltmpi -n 2 prog.x
date

echo "DONE"
ltmend
[wly0m@thunder1 first]$
```



Let's do it together

Step # 7: Execute procedure

```
[wly0m@thunder1 first]$ cat rwave
ltmbegin -n 2

##### COMPILE PROGRAM:
rm prog.x
mpif90 -o prog.x -maxcontin=25 -w -V wave.f

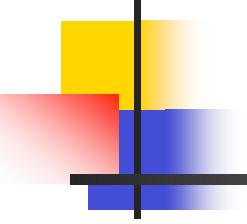
##### RUN
cp -p prog.x $HOME/rhome/first
date
ltmpi -n 2 prog.x
date

echo "DONE"
ltmend

[wly0m@thunder1 first]$ ltmsuper
[wly0m@thunder1 first]$ rwave &
[wly0m@thunder1 first]$
```

```
[wly0m@thunder1 first]$ rwave &
[1] 7224
[wly0m@thunder1 first]$ -c option specified twice, continuing
NAGWare Fortran 95 compiler Release 4.2(464)
Copyright 1990-2002 The Numerical Algorithms Group Ltd., Oxford, U.K.
f95comp version is 5.0(322)
NAGWare Fortran 95 compiler Release 4.2(464)
Copyright 1990-2002 The Numerical Algorithms Group Ltd., Oxford, U.K.
Fri Feb 4 10:41:20 EST 2005
per_node=2 residual=0 no_of_nodes=2 tasks=4
command = /home/wly0m/.mpi_gm/mpich_home/bin/mpirun -np 4 -
machinefile /home/ltm//wly0m/nodelist-1107807111.416753 prog.x
2: Wave Program running
2IL: first = 501 npoints = 250
3: Wave Program running
3IL: first = 751 npoints = 250
1: Wave Program running
1IL: first = 251 npoints = 250
0: Wave Program running
0IL: first = 1 npoints = 250
Fri Feb 4 10:41:21 EST 2005
DONE

[1]+ Done rwave
[wly0m@thunder1 first]$
```



Let's do it together

Step # 7: Execute procedure

```
[wly0m@thunder1 first]$ cat rwave
ltmbegin -n 2

##### COMPILE PROGRAM:
rm prog.x
mpif90 -o prog.x -maxcontin=25 -w -V wave.f

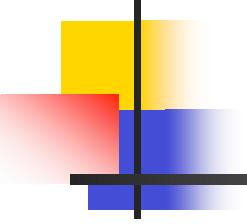
##### RUN
cp -p prog.x $HOME/rhome/first
date
ltmpi -n 2 prog.x
date

echo "DONE"
ltmend

[wly0m@thunder1 first]$ ltmsuper
[wly0m@thunder1 first]$ rwave &
[wly0m@thunder1 first]$ ltmterminate
```

```
[wly0m@thunder1 first]$ rwave &
[1] 7224
[wly0m@thunder1 first]$ -c option specified twice, continuing
NAGWare Fortran 95 compiler Release 4.2(464)
Copyright 1990-2002 The Numerical Algorithms Group Ltd., Oxford, U.K.
f95comp version is 5.0(322)
NAGWare Fortran 95 compiler Release 4.2(464)
Copyright 1990-2002 The Numerical Algorithms Group Ltd., Oxford, U.K.
Fri Feb 4 10:41:20 EST 2005
per_node=2 residual=0 no_of_nodes=2 tasks=4
command = /home/wly0m/.mpi_gm/mpich_home/bin/mpirun -np 4 -
machinefile /home/ltm//wly0m/nodelist-1107807111.416753 prog.x
2: Wave Program running
2IL: first = 501 npoints = 250
3: Wave Program running
3IL: first = 751 npoints = 250
1: Wave Program running
1IL: first = 251 npoints = 250
0: Wave Program running
0IL: first = 1 npoints = 250
Fri Feb 4 10:41:21 EST 2005
DONE

[1]+ Done rwave
[wly0m@thunder1 first]$
```



Let's do it together

Step # 7: Execute procedure

```
[wly0m@thunder1 first]$ cat rwave
ltmbegin -n 2

##### COMPILE PROGRAM:
rm prog.x
mpif90 -o prog.x -maxcontin=25 -w -V wave.f

##### RUN
cp -p prog.x $HOME/rhome/first
date
ltmpi -n 2 prog.x
date

echo "DONE"
ltmend

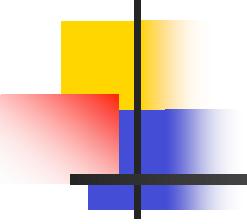
[wly0m@thunder1 first]$ ltmsuper
[wly0m@thunder1 first]$ rwave &
[wly0m@thunder1 first]$
```

7/27/05

```
[wly0m@thunder1 first]$ rwave &
[1] 7224
[wly0m@thunder1 first]$ -c option specified twice, continuing
NAGWare Fortran 95 compiler Release 4.2(464)
Copyright 1990-2002 The Numerical Algorithms Group Ltd., Oxford, U.K.
f95comp version is 5.0(322)
NAGWare Fortran 95 compiler Release 4.2(464)
Copyright 1990-2002 The Numerical Algorithms Group Ltd., Oxford, U.K.
Fri Feb 4 10:41:20 EST 2005
per_node=2 residual=0 no_of_nodes=2 tasks=4
command = /home/wly0m/.mpi_gm/mpich_home/bin/mpirun -np 4 -
machinefile /home/ltm//wly0m/nodelist-1107807111.416753 prog.x
2: Wave Program running
2IL: first = 501 npoints = 250
3: Wave Program running
3IL: first = 751 npoints = 250
1: Wave Program running
1IL: first = 251 npoints = 250
0: Wave Program running
0IL: first = 1 npoints = 250
Fri Feb 4 10:41:21 EST 2005
DONE

[1]+ Done rwave
[wly0m@thunder1 first]$
```

55



Let's do it together

Step # 7: Execute procedure

```
[wly0m@thunder1 first]$ cat rwave
ltmbegin -n 2

##### COMPILE PROGRAM:
rm prog.x
mpif90 -o prog.x -maxcontin=25 -w -V wave.f

##### RUN
cp -p prog.x $HOME/rhome/first
date
ltmpi -n 2 prog.x
date

echo "DONE"
ltmend

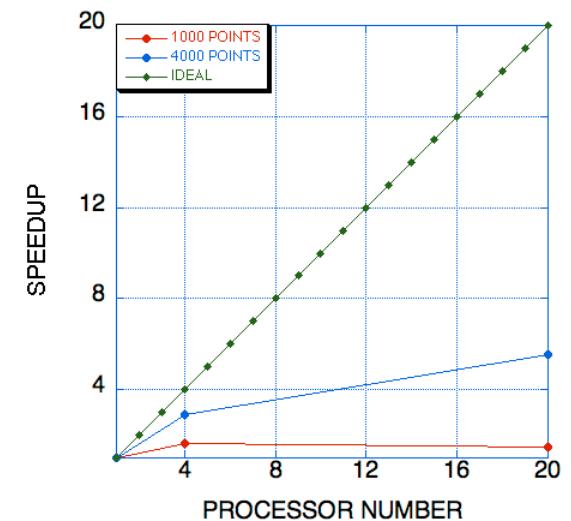
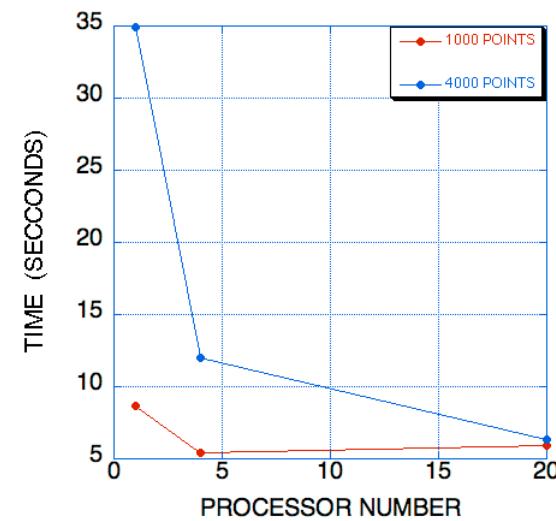
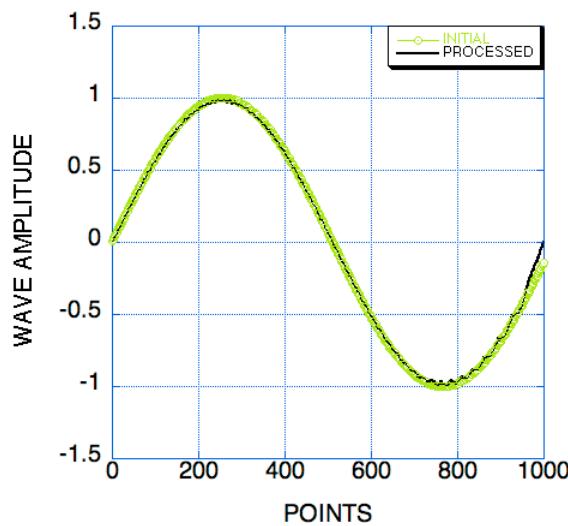
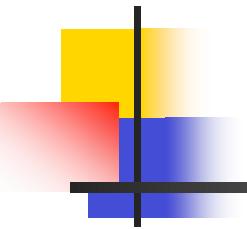
[wly0m@thunder1 first]$ ltmsuper
[wly0m@thunder1 first]$ rwave &
[wly0m@thunder1 first]$ ltmterminate
```

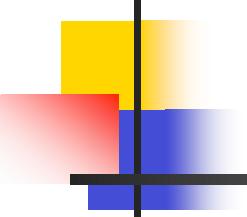
```
[wly0m@thunder1 first]$ rwave &
[1] 7224
[wly0m@thunder1 first]$ -c option specified twice, continuing
NAGWare Fortran 95 compiler Release 4.2(464)
Copyright 1990-2002 The Numerical Algorithms Group Ltd., Oxford, U.K.
f95comp version is 5.0(322)
NAGWare Fortran 95 compiler Release 4.2(464)
Copyright 1990-2002 The Numerical Algorithms Group Ltd., Oxford, U.K.
Fri Feb 4 10:41:20 EST 2005
per_node=2 residual=0 no_of_nodes=2 tasks=4
command = /home/wly0m/.mpi_gm/mpich_home/bin/mpirun -np 4 -
machinefile /home/ltm//wly0m/nodelist-1107807111.416753 prog.x
2: Wave Program running
2IL: first = 501 npoints = 250
3: Wave Program running
3IL: first = 751 npoints = 250
1: Wave Program running
1IL: first = 251 npoints = 250
0: Wave Program running
0IL: first = 1 npoints = 250
Fri Feb 4 10:41:21 EST 2005
DONE

[1]+ Done rwave
[wly0m@thunder1 first]$ ltmterminate
```

Let's do it together

Step # 8: Analyze the output

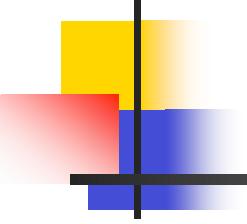




Support

James L Green
Chief, Space Science Data Operations Office
November 22, 2004

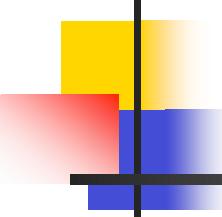
- **THUNDERHEAD - Open to everyone in SSD!**
 - Developed for “Grand Challenges” in NASA sciences
 - Program ended October 1, 2004
 - Partial funding for support in FY05 found
 - System Admin (1.5) - short about \$100K
 - Algorithm support and website development (1.5 FTE/AETD)
 - Some Hardware/software - short about \$25K
 - System will be managed by the NEW IT Services organization
- **THUNDERHEAD viewed as a “stepping stone” to Project Columbia computing**



Support

Support for THUNDERHEAD

- THUNDERHEAD will be needed for:
 - Code development
 - Visualization
 - Local modeling with quick turn around
 - Without future \$ support system will be turned off
- Need to start planning for proposing operational costs for THUNDERHEAD in upcoming ROSES NRA call in Jan.
 - The more users the less the proposed cost
 - Cost per user/group TBD
- Total costs depends on levels of support
 - System Administration, User support, web maintenance....etc.
- User group need to be defined to set policy usage and services needed before costs can be determined



References

- William Gropp et al., 1999: ***Using MPI. Portable Parallel Programming with the Message Passing Interface.*** The MIT Press, Cambridge Massachusetts, London England
- Marc Snir et al., 2000: ***MPI-The Complete Reference.*** The MIT Press, Cambridge Massachusetts, London England
- ***Thunderhead Web Site*** <http://thunderhead.gsfc.nasa.gov/>
- ***Cornell Theory Center*** <http://www.tc.cornell.edu/>
- ***'CRAY T3E Applications programming'*** Cray Research